Criptosistema

RSA

Prof. Dr. Iaçanã Ianiski Weber

Confiabilidade e Segurança de Software

98G08-4

Agradecimentos especiais ao Prof. Avelino Zorzo e aos Autores Christof Paar e Jan Pelzl pelo material.

Índice

1. O Criptosistema RSA
2. Aspectos de Implementação
3. Encontrar Números Primos Grandes
4. Ataques e Contramedidas
5. Conclusão
6. Exercícios

Índice

1. O Criptosistema RSA
2. Aspectos de Implementação
3. Encontrar Números Primos Grandes
4. Ataques e Contramedidas
5. Conclusão
6. Exercícios

O Criptossistema RSA

- Martin Hellman e Whitfield Diffie publicaram seu artigo sobre public-key em 1976.
- Ronald Rivest, Adi Shamir e Leonard Adleman propuseram o criptossistema assimétrico RSA em 1977.
- Até hoje, o RSA é o criptossistema assimétrico mais utilizado, embora a criptografia de curva elíptica (ECC) esteja se tornando cada vez mais popular.
- O RSA é usado principalmente para duas aplicações:
  - Troca de chaves para uso de cifras simétricas.
  - Assinaturas digitais.

Cifragem e Decifragem

- As operações RSA são realizadas sobre o anel de inteiros Z_n (i.e., aritmética módulo n), onde $n = p \cdot q$, com p, q sendo primos grandes.
- Cifragem e decifragem são simplesmente exponenciações no anel.

Definição

Dada a public key $(n, e) = k_{\text{pub}}$ e a private key $d = k_{\text{pr}}$ escrevemos:

$$y = e_{k_{pub}}(x) \equiv x^e \pmod{n}$$
$$x = d_{k_{pr}}(y) \equiv y^d \pmod{n}$$

onde $x, y \in Z_n$.
Chamamos $e_{k_{\text{pub}}}()$ a operação de cifragem e $d_{k_{\text{pr}}}()$ a operação de decifragem.

- Na prática x, y, n e d são números inteiros muito longos (≥ 1024 bits).
- A segurança do esquema reside no fato de que é difícil derivar o 'expoente privado' d dada a public-key (n, e).

Geração de Keys

- Como todos os esquemas assimétricos, o RSA possui uma fase de set-up durante a qual as private e public keys são computadas.

Algoritmo: Geração de Keys RSA

Saída: public key $k_{\text{pub}} = (n, e)$ e private key $k_{\text{pr}} = d$

1. Escolha dois primos grandes p, q.
2. Compute $n = p \cdot q$.
3. Compute $\Phi(n) = (p-1)(q-1)$.
4. Selecione o expoente público $e \in \{1, 2, \ldots, \Phi(n) - 1\}$ tal que $mdc(e, \Phi(n)) = 1$.
5. Computa uma chave privada d tal que $d \cdot e \equiv 1 \pmod{\Phi(n)}$.
6. **RETORNE** $k_{\text{pub}} = (n, e)$, $k_{\text{pr}} = d$.

Observações:

- Escolher dois primos grandes e distintos p, q (no Passo 1) não é trivial.
- $mdc(e, \Phi(n)) = 1$ assegura que e possui um inverso e, assim, que sempre existe uma chave privada d.

Exemplo: RSA com números pequenos

ALICE

Message $x = 4$

BOB

1. Choose $p = 3$ and $q = 11$
2. Compute $n = p * q = 33$
3. $\Phi(n) = (3-1) * (11-1) = 20$
4. Choose $e = 3$
5. $d \equiv e^{-1} \equiv 7 \bmod 20$

$K_{pub} = (33,3)$ ←

$y = x^e \equiv 4^3 \equiv 31 \bmod 33$

$y = 31$ →

$y^d = 31^7 \equiv 4 = x \bmod 33$

Índice

1. O Criptosistema RSA
2. Aspectos de Implementação
3. Encontrar Números Primos Grandes
4. Ataques e Contramedidas
5. Conclusão
6. Exercícios

Aspectos de Implementação

- O criptossistema RSA usa apenas uma operação aritmética (exponenciação modular), o que o torna conceitualmente um esquema assimétrico simples.
- Embora conceitualmente simples, devido ao uso de números muito longos, o RSA é ordens de magnitude mais lento que esquemas simétricos, como o AES.
- Ao implementar o RSA (especialmente em um dispositivo com restrições, como smartcards ou celulares), atenção especial deve ser dada à escolha correta de algoritmos aritméticos.
- O algoritmo *square-and-multiply* permite exponenciação rápida, mesmo com números muito longos. . .

Square-and-Multiply

- Princípio básico: Varrer os bits do expoente da esquerda para a direita e eleve ao quadrado/multiplique o operando de acordo.

Algoritmo: Square-and-Multiply para $x^H \pmod{n}$

Entrada: Expoente H, elemento base x, Módulo n.
Saída: $y = x^H \pmod{n}$.

1. Determine a representação binária $H = (h_t, h_{t-1}, \ldots, h_0)_2$. (Assuma $H > 0$ e que $h_t = 1$ é o bit mais significativo).
2. $y \leftarrow x$.
3. **PARA** i de $t - 1$ **ATÉ** 0 **FAÇA**
4. $\quad y \leftarrow y^2 \pmod{n}$.
5. $\quad$ **SE** $h_i = 1$ **ENTÃO**
6. $\quad\quad y \leftarrow y \cdot x \pmod{n}$.
7. **RETORNE** y.

- **Regra**: Elevar ao quadrado em cada iteração (conforme a operação $y \leftarrow y^2 \pmod{n}$ no laço) e multiplicar o resultado atual por x se o bit do expoente $h_i = 1$ (conforme a operação $y \leftarrow y \cdot x \pmod{n}$).
- A redução modular após cada passo mantém o operando y pequeno.

Exemplo: *Square-and-Multiply*

- Calcula x^{26} sem redução modular.
- Representação binária do expoente:
  $26 = (1, 1, 0, 1, 0)_2 = (h_4, h_3, h_2, h_1, h_0)_2$.

Passo	Valor computado	Expoente binário	Op
1	$y = x = x^1$	$(1)_2$	-
1a	$(x^1)^2 = x^2$	$(10)_2$	SQ
1b	$x^2 \cdot x = x^3$	$(11)_2$	MUL
2a	$(x^3)^2 = x^6$	$(110)_2$	SQ
2b	-	$(110)_2$	-
3a	$(x^6)^2 = x^{12}$	$(1100)_2$	SQ
3b	$x^{12} \cdot x = x^{13}$	$(1101)_2$	MUL
4a	$(x^{13})^2 = x^{26}$	$(11010)_2$	SQ
4b	-	$(11010)_2$	-

- Observe como o expoente evolui para $x^{26} = x^{(11010)_2}$.

Complexidade do Algoritmo *Square-and-Multiply*

- O algoritmo *square-and-multiply* possui complexidade logarítmica, ou seja, seu tempo de execução é proporcional ao comprimento em bits (em vez do valor absoluto) do expoente.
- Dado um expoente com $t+1$ bits $H = (h_t, h_{t-1}, \ldots, h_0)_2$ com $h_t = 1$, necessitamos das seguintes operações:
  - Número de SQ (elevações ao quadrado) $= t$
  - Número médio de MUL (multiplicações) $= 0.5t$
  - Complexidade total: Nº SQ + Nº MUL $= 1.5t$
- Expoentes são frequentemente escolhidos aleatoriamente, então $1.5t$ é uma boa estimativa para o número médio de operações.
- Note que cada SQ (elevação ao quadrado) e cada MUL (multiplicação) é uma operação com números muito longos, e.g., inteiros de 2048 bits.

Técnicas de Aceleração

- A exponenciação modular é computacionalmente intensiva.
- Mesmo com o algoritmo square-and-multiply, o RSA pode ser bastante lento em dispositivos com restrições, como smart cards.
- Alguns truques importantes:
  - Expoente público *e* curto
  - Teorema Chinês do Resto (CRT)
  - Exponenciação com pré-computação (*não abordado aqui*)

Criptografia rápida com expoente público pequeno

- Escolher um expoente público e pequeno não enfraquece a segurança do RSA
- Um expoente público pequeno melhora significativamente a velocidade da criptografia RSA

Chave Pública (e)	**e como string binária**	**#mul + #quad**
3	$(11)_2$	$1+1=2$
17	$(10001)_2$	$4+1=5$
$2^{16}+1$	$(1000000000000001)_2$	$16+1=17$

- Este é um truque comumente usado (ex.: SSL/TLS, etc.) e torna o RSA o esquema assimétrico mais rápido em relação ao processo de encriptação!

Decriptografia Rápida com o Teorema Chinês do Resto

- A escolha de uma chave privada d pequena resulta em fraquezas de segurança!
  - De fato, d deve ter pelo menos $0.3t$ bits, onde t é o comprimento em bits do módulo n.
- No entanto, o **Teorema Chinês do Resto (CRT)** pode ser usado para acelerar (de certa forma) a exponenciação com a chave privada d.
- Baseado no **CRT**, podemos substituir a computação de:

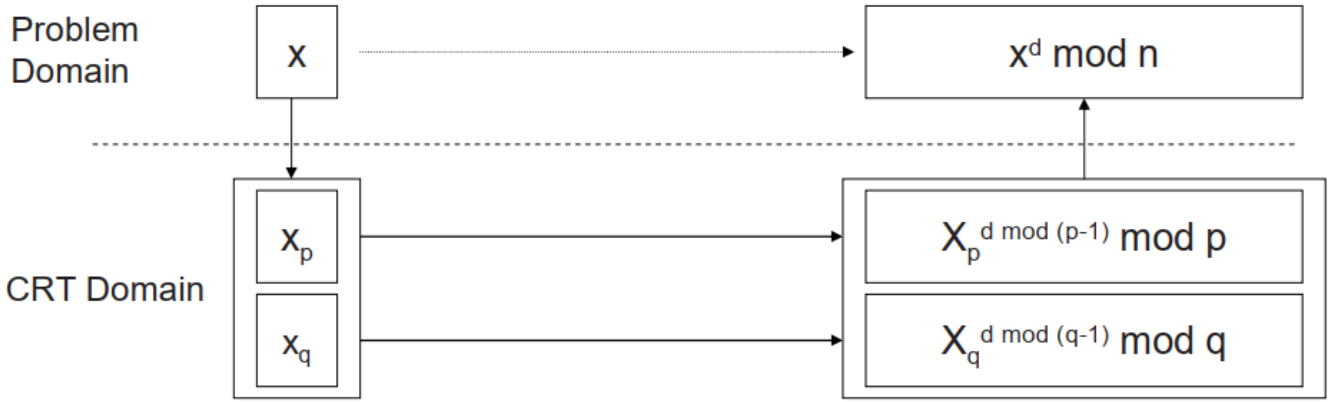
$$x^d \pmod{\Phi(n)} \pmod{n}$$

por duas computações:

$$x^d \pmod{p-1} \pmod{p} \quad \text{e} \quad x^d \pmod{q-1} \pmod{q}$$

onde q e p são "pequenos"em comparação com n.

Princípio Básico da Exponenciação Baseada em CRT



- O **CRT** envolve três etapas distintas:
  (1) Transformação do operando para o domínio CRT
  (2) Exponenciação modular no domínio CRT
  (3) Transformação inversa para o domínio do problema

- Essas etapas são equivalentes a uma exponenciação modular no domínio do problema.

Complexidade do CRT

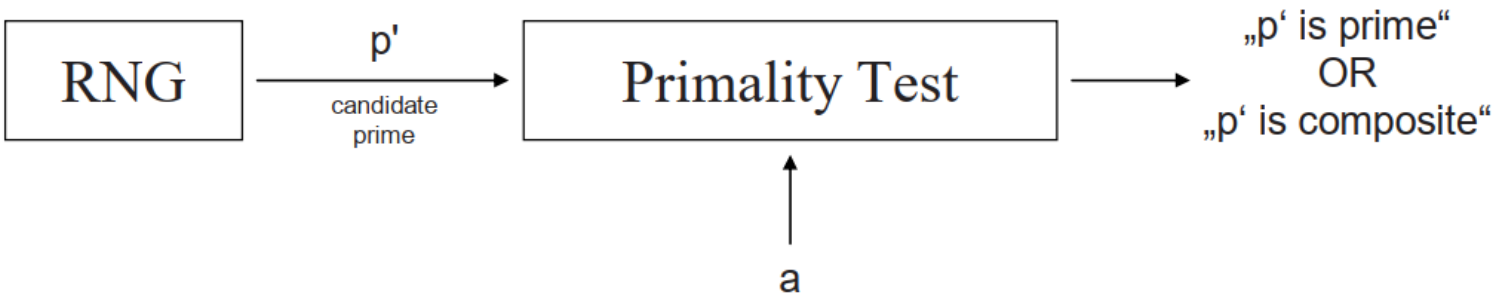
- Ignoramos as etapas de transformação e transformação inversa, pois seus custos podem ser negligenciados sob premissas razoáveis.
- Assumindo que n tem $t+1$ bits, tanto p quanto q têm aproximadamente $t/2$ bits de comprimento.
- A complexidade é determinada pelas duas exponenciações no domínio CRT. Os operandos têm apenas $t/2$ bits de comprimento. Para as exponenciações, usamos o algoritmo *square-and-multiply*:
  - # quadrados (uma exp.): $\#\text{QUAD} = 0.5t$
  - # mult. médias (uma exp.): $\#\text{MULT} = 0.25t$
  - Complexidade total: $2 * (\#\text{MULT} + \#\text{QUAD}) = 1.5t$
- Isso parece o mesmo que exponenciações regulares, mas como os operandos têm metade do comprimento de bits em comparação com exponenciações regulares, cada operação (ou seja, multiplicação e quadratura) é **4 vezes mais rápida!**
- Portanto, o CRT é **4 vezes mais rápido** que a exponenciação direta.

Índice

1. O Criptosistema RSA
2. Aspectos de Implementação
3. Encontrar Números Primos Grandes
4. Ataques e Contramedidas
5. Conclusão
6. Exercícios

Encontrando Primos Grandes

- A geração de chaves para o RSA requer encontrar dois primos grandes p e q tal que $n = p * q$ seja suficientemente grande.
- O tamanho de p e q é tipicamente metade do tamanho desejado de n.
- Para encontrar primos, inteiros aleatórios são gerados e testados para primalidade:



- O gerador de números aleatórios (RNG) deve ser não-previsível, caso contrário um atacante poderia adivinhar a fatoração de n.

Testes de Primalidade

- Fatorar p e q para testar a primalidade geralmente não é factível.
- No entanto, não estamos interessados na fatoração, apenas queremos saber se p e q são compostos ou não.
- Testes de primalidade típicos são probabilísticos, ou seja, não são 100% precisos, mas sua saída é correta com probabilidade muito alta.
- Um teste probabilístico tem duas saídas:
  - p' é composto – sempre verdadeiro
  - p' é primo – verdadeiro apenas com uma certa probabilidade
- Entre os testes de primalidade bem conhecidos estão os seguintes:
  - Teste de Primalidade de Fermat
  - Teste de Primalidade de Miller-Rabin

Teste de Primalidade de Fermat

- Ideia básica: O Pequeno Teorema de Fermat é válido para todos os primos, i.e., se um número p' é encontrado para o qual $a^{p'-1} \not\equiv 1 \pmod{p'}$, ele não é primo.

Algoritmo: Teste de Primalidade de Fermat

Entrada: Candidato a primo p', parâmetro de segurança s
Saída: "p' é composto" ou "p' é provavelmente primo"

1. **PARA** $i = 1$ **ATÉ** s
2. Escolha $a \in \{2, 3, \ldots, p' - 2\}$ aleatoriamente
3. **SE** $a^{p'-1} \not\equiv 1 \pmod{p'}$ **ENTÃO**
4. **RETORNE** "p' é composto"
5. **RETORNE** "p' é provavelmente primo"

- Para certos números ("números de Carmichael"), este teste retorna "p' é provavelmente primo" frequentemente – embora esses números sejam compostos.

Teorema para o Teste de Miller-Rabin

- O Teste de Miller-Rabin, mais poderoso, é baseado no seguinte teorema:

Teorema

Dada a decomposição de um candidato a primo ímpar p'

$$p' - 1 = 2^u \cdot r$$

onde r é ímpar.
Se pudermos encontrar um inteiro a tal que

$$a^r \not\equiv 1 \pmod{p'} \quad \text{e} \quad a^{2^j \cdot r} \not\equiv p' - 1 \pmod{p'}$$

Para todo $j \in \{0, 1, \ldots, u-1\}$, então p' é composto.
Caso contrário, ele é provavelmente primo.

- Este teorema pode ser transformado em um algoritmo.

Teste de Primalidade de Miller-Rabin

Algoritmo: Teste de Primalidade de Miller-Rabin

Entrada: Candidato a primo p' com $p' - 1 = 2^u \cdot r$, parâmetro de segurança s

Saída: "p' é composto" ou "p' é provavelmente primo"

1. **PARA** $i = 1$ **ATÉ** s
2. Escolha $a \in \{2, 3, \ldots, p' - 2\}$ aleatoriamente
3. $z \equiv a^r \pmod{p'}$
4. **SE** $z \neq 1$ **E** $z \neq p' - 1$ **ENTÃO**
5. **PARA** $j = 1$ **ATÉ** $u - 1$
6. $z \equiv z^2 \pmod{p'}$
7. **SE** $z = 1$ **ENTÃO**
8. **RETORNE** "p' é composto"
9. **SE** $z \neq p' - 1$ **ENTÃO**
10. **RETORNE** "p' é composto"
11. **RETORNE** "p' é provavelmente primo"

Índice

1. O Criptosistema RSA
2. Aspectos de Implementação
3. Encontrar Números Primos Grandes
4. Ataques e Contramedidas
5. Conclusão
6. Exercícios

Ataques e Contramedidas 1/3

- Existem dois tipos distintos de ataques a criptossistemas.
- **Ataques analíticos** tentam quebrar a estrutura matemática do problema subjacente do RSA.
- **Ataques de implementação** tentam atacar uma implementação do mundo real explorando fraquezas inerentes à forma como o RSA é implementado em software ou hardware.

Ataques e Contramedidas 2/3

O RSA está tipicamente exposto a estes vetores de ataque analíticos.

- **Ataques matemáticos**
  - O ataque mais conhecido é a fatoração de n a fim de obter $\Phi(n)$.
  - Podem ser prevenidos utilizando um módulo n suficientemente grande.
  - O recorde atual de fatoração é de 829 bits. Portanto, é recomendado que n possua um comprimento em bits maior que 2048.
- **Ataques de protocolo**
  - Exploram a maleabilidade do RSA, i.e., a propriedade de que um ciphertext pode ser transformado em outro ciphertext que decifra para um plaintext relacionado – sem conhecer a private key.
  - Podem ser prevenidos por padding adequado.

Ataques e Contramedidas 3/3

- Ataques de implementação podem ser um dos seguintes:
  - **Análise de canal lateral**
    - Explora o vazamento físico da implementação RSA (e.g., consumo de energia, emanação EM, etc.).
  - **Ataques de injeção de falhas**
    - Induzir falhas no dispositivo enquanto o CRT é executado pode levar a um vazamento completo da private key.

Índice

1. O Criptosistema RSA
2. Aspectos de Implementação
3. Encontrar Números Primos Grandes
4. Ataques e Contramedidas
5. Conclusão
6. Exercícios

Lições Aprendidas

- O RSA é o criptossistema de chave pública mais amplamente utilizado.
- O RSA é usado principalmente para transporte de chaves (key transport) e assinaturas digitais.
- Uma chave pública 'e' pode ser um inteiro curto, já a private key d precisa ter o comprimento total do módulo n.
- O RSA baseia-se no fato de que é difícil fatorar n.
- O recorde de fatoração é de 829 bits (RSA-250). Chaves de 1024 bits não são mais consideradas seguras para uso a longo prazo. Recomenda-se, no mínimo, chaves RSA de 2048 bits, e chaves de 3072 bits (NIST level 1) ou superiores (e.g., 7680 bits, 15360 bits para NIST level 2 e 3) para maior longevidade ou para novos sistemas.
- Uma implementação ingênua do RSA permite diversos ataques.

Índice

1. O Criptosistema RSA
2. Aspectos de Implementação
3. Encontrar Números Primos Grandes
4. Ataques e Contramedidas
5. Conclusão
6. Exercícios

Exercício: Parâmetros RSA

1. Sejam os primos $p = 41$ e $q = 17$ dados como parâmetros de configuração para o RSA.

1. Qual dos parâmetros $e_1 = 32, e_2 = 49$ é um expoente RSA válido? Justifique sua escolha.

2. Calcule a private key $k_{pr} = (p, q, d)$ correspondente. Use o algoritmo de Euclides estendido para a inversão e mostre cada passo do cálculo.

Exercício: Exponenciação Modular com Square-and-Multiply

2. A computação eficiente da exponenciação modular é fundamental para o uso prático do RSA. Calcule as seguintes exponenciações $x^e \pmod{m}$ usando o algoritmo de elevação ao quadrado e multiplicação:

1. $x = 2, e = 79, m = 101$
2. $x = 3, e = 197, m = 101$
3. $x = 5, e = 54, m = 151$
4. $x = 8, e = 127, m = 151$

Após cada passo de iteração, mostre o expoente do resultado intermediário em notação binária.

Exercício: Criptografia e Decriptografia RSA

3. Cifre e decifre utilizando o algoritmo RSA com os seguintes parâmetros de sistema:

1. $p = 3; \quad q = 11; \quad d = 7; \quad x = 5$
2. $p = 5; \quad q = 11; \quad e = 3; \quad x = 9$

Utilize apenas uma calculadora simples.